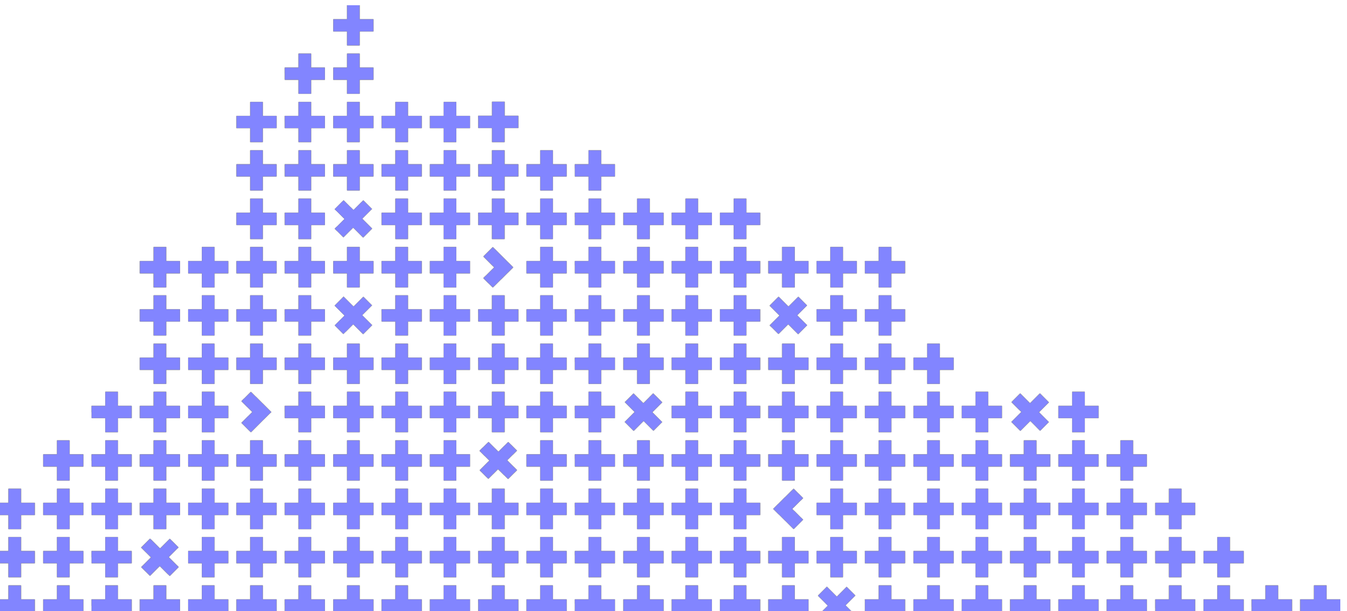
PostgreSQL: a Journey from 0TB to 40TB in 4 Years

Jordan PITTIER

gorgias



High Load ++ Armenia

Co-organizer

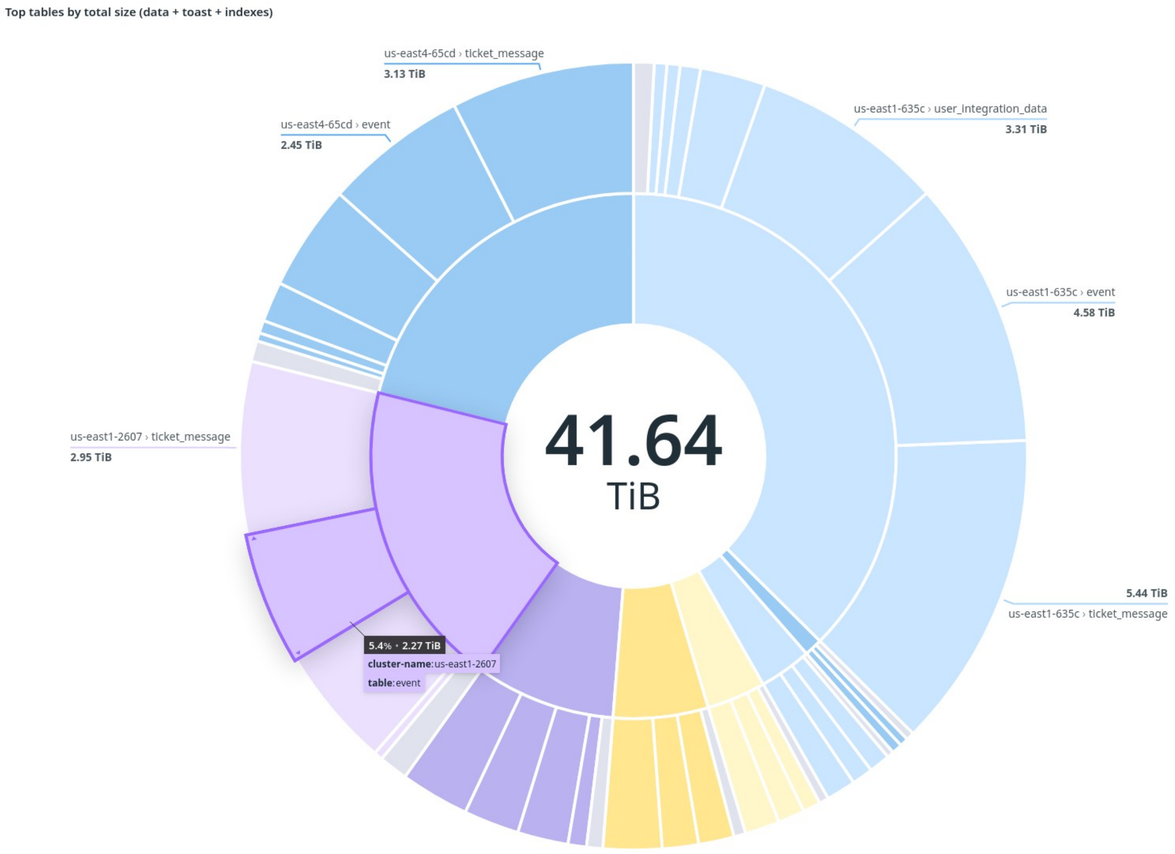
Yandex

Agenda

- Mistakes we’ve made as Developers and how we’ve fixed them
- Mistakes we’ve made as Operations and how we’ve fixed them

About me

- I am not a DBA
- I am not even a DB Reliability Engineer (yes, that’s a job!)
- I am a SRE with a backend SWE background, but I manage this:



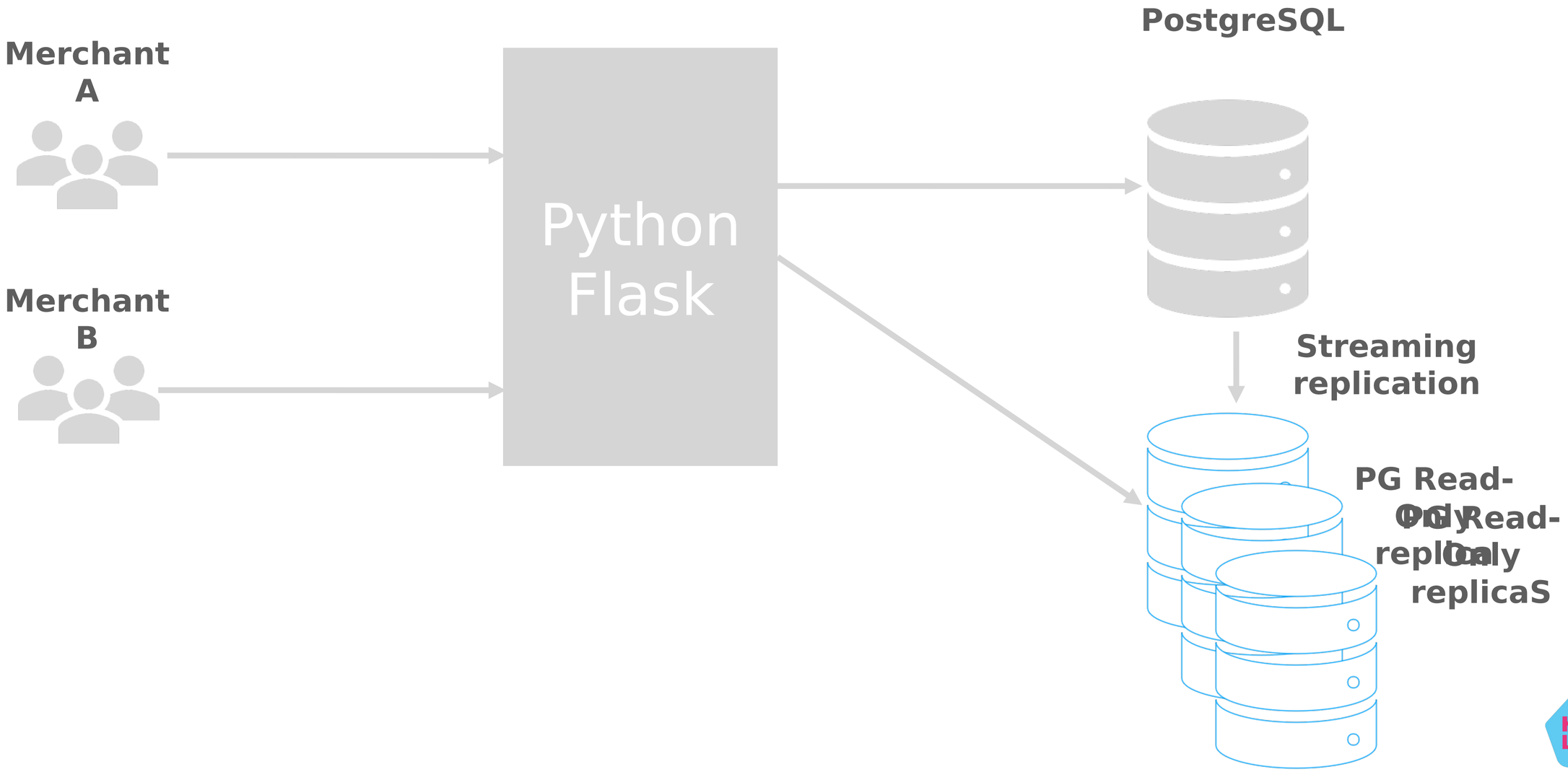
About Gorgias (Hi Boss!)

- Sales Pitch:
  * We build an integrated helpdesk for e-commerce brands, making it easy to deliver personalized support and automation across multiple channels. Connect all your business and social apps, and turn customer support into a revenue-generating activity!

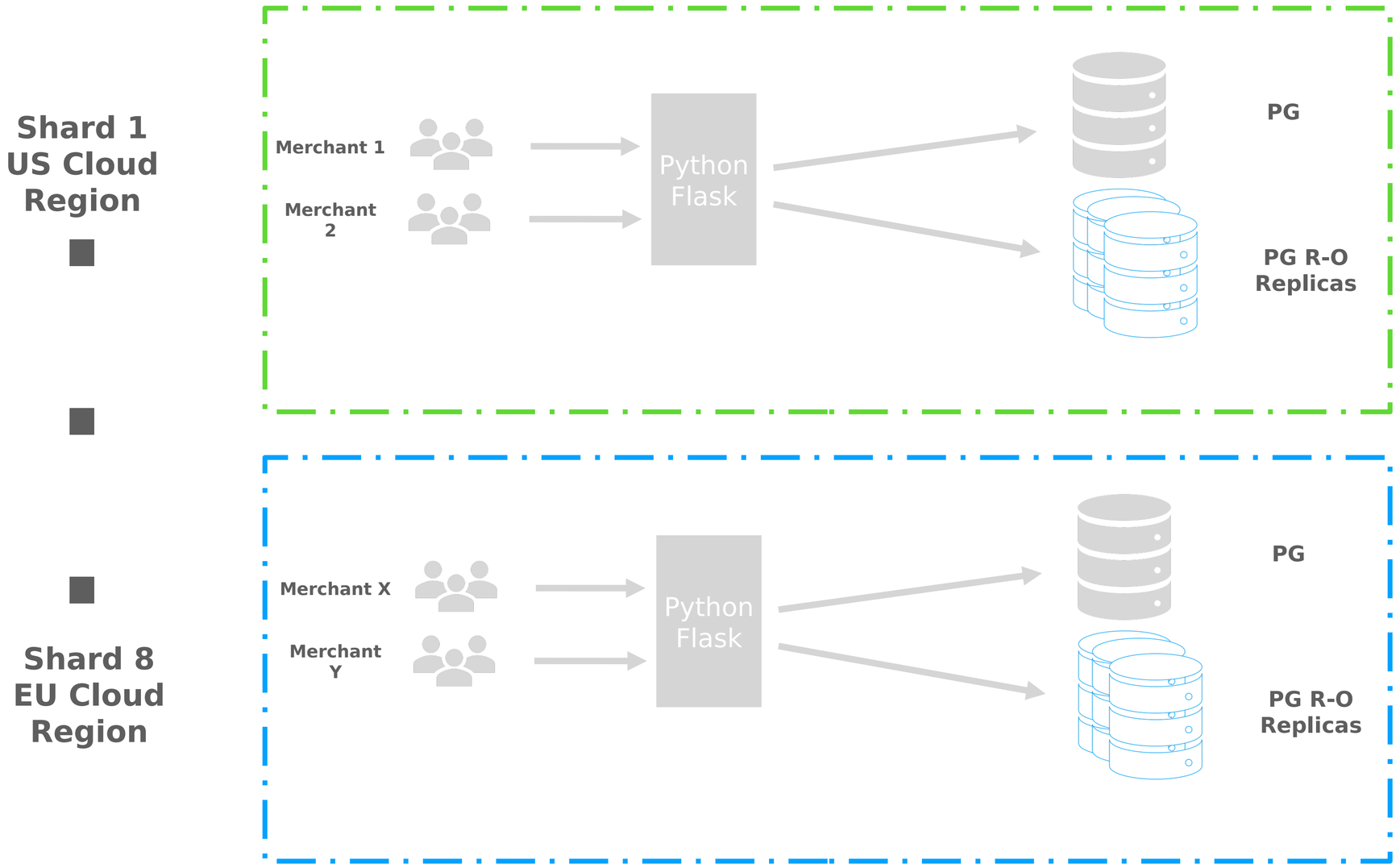
  * Gorgias empowers 10,000+ online merchants like SteveMadden, PrincessPolly or MarineLayer to provide the best possible experience to their customers.

- TLDR: Provide supercharged mailboxes for online merchants, as SaaS

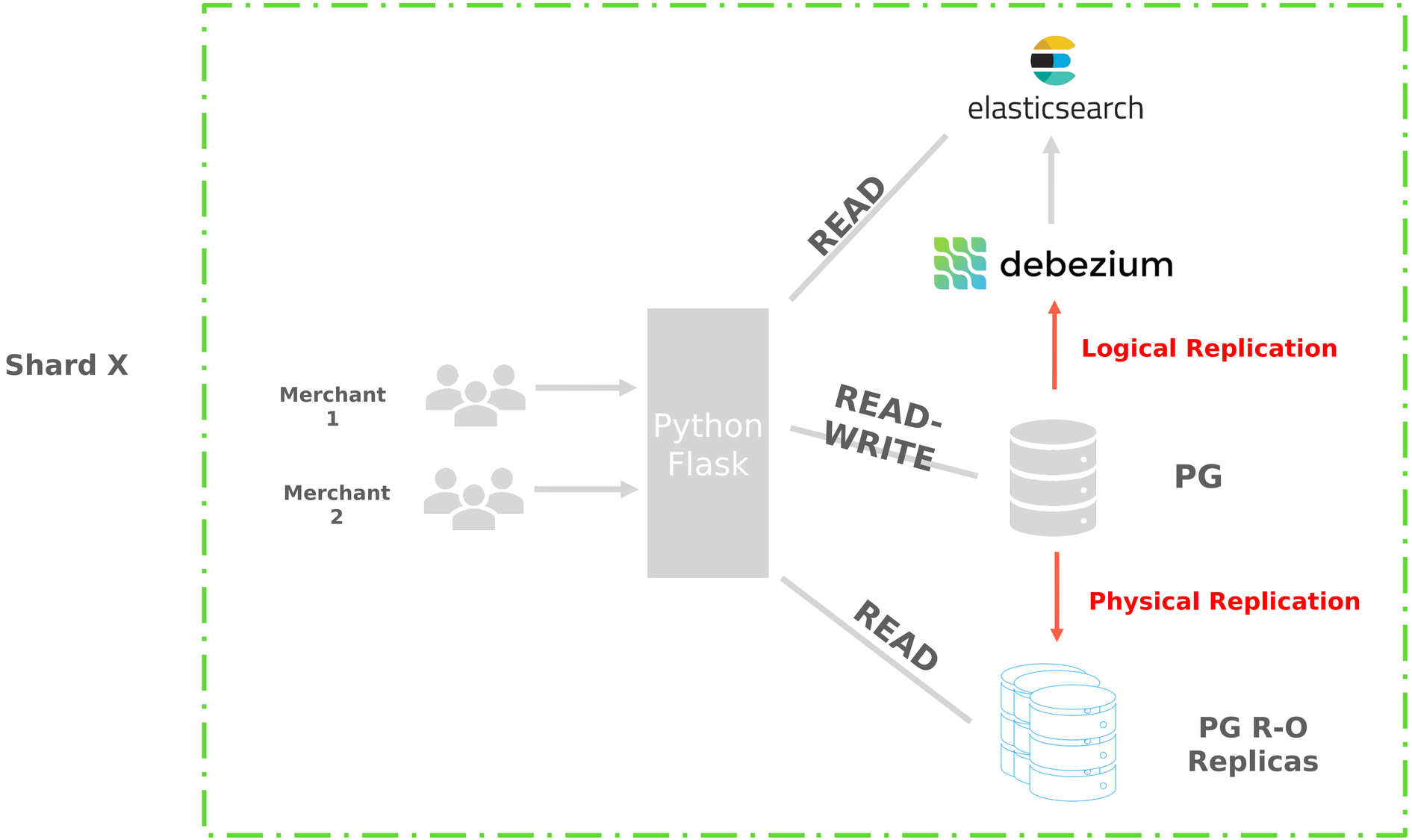
In the beginning



Multi Shards Architecture

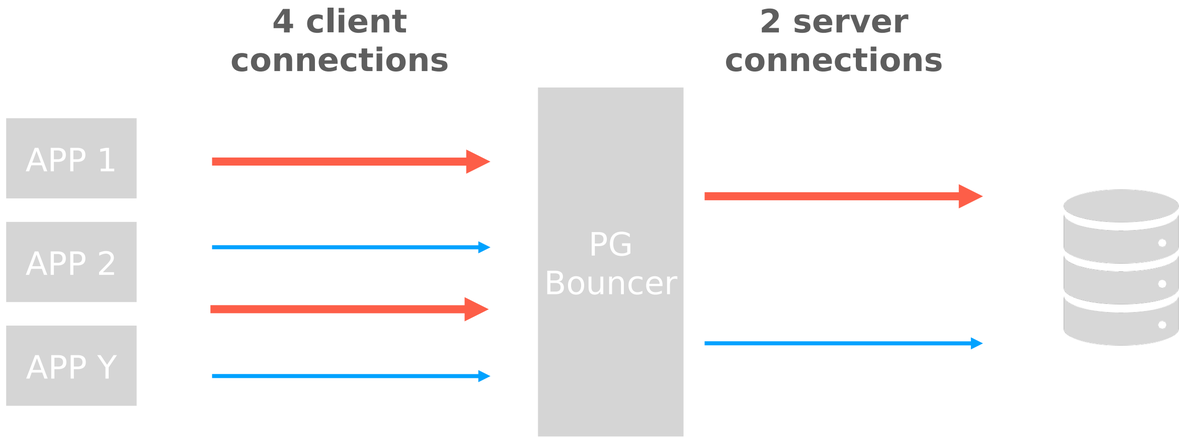


Using ElasticSearch for Full Text Search



Issue 1: PG Sessions in idle-in-tx

- PG connections are expensive in CPU and memory
- max_connections in postgresql.conf set to a few hundreds
- With PGBouncer, the same PG connection can be used by several clients … but not at the same time!



- Connections in the middle of a TX can’t be reused
- Problem: what if the client (i.e the app) forgets or takes a long time to commit/rollback ? At 3000 TX/s => quick connection starvation

Idle-in-tx (2)

- Typically happens when

```
BEGIN
SELECT * from messages;
For each messages {
    heavy Python processing and/or slow network calls
}
COMMIT
```

- Solutions?

1. Fix the application code: not so easy
2. Set idle_in_transaction_session_timeout = “10s” (ALTER SYSTEM|DATABASE|USER)

Issue 2: Vacuuming Large Tables

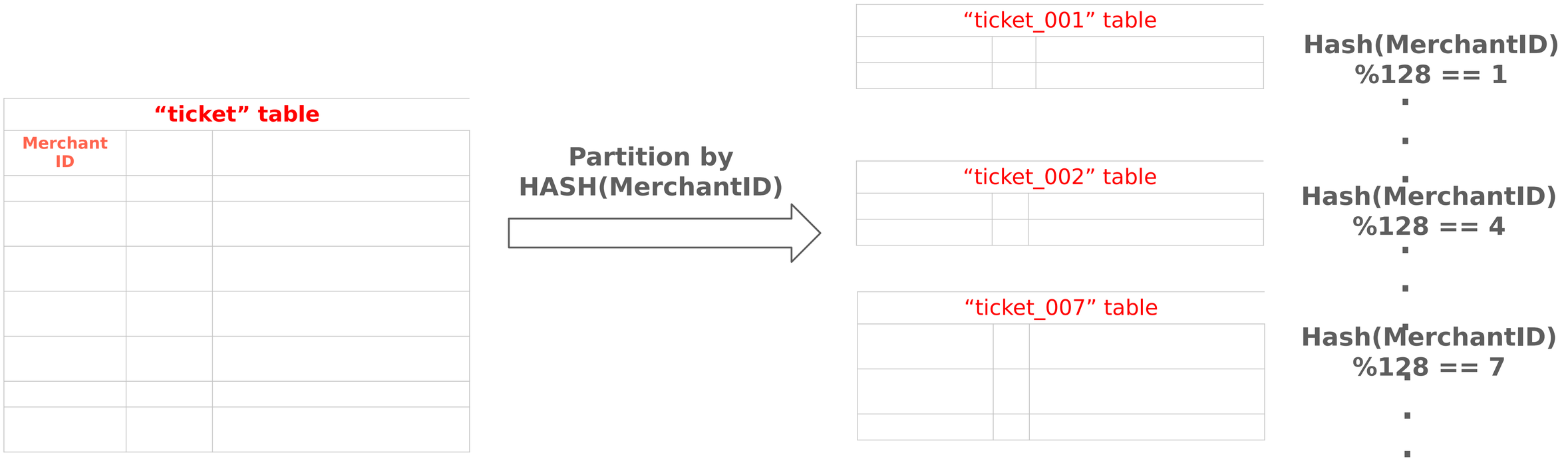
- Vacuum is a maintenance operation that removes dead rows from the table and index files.
- Vacuum is critical to maintain consistent query performance

- Vacuum operates in 3 phases:

1. Scan the table: look for the dead rows and save their “location” in a in-memory buffer
2. Cleanup indexes: delete elements that point to the location of dead rows (using the above buffer)
3. Cleanup the table files by making the dead rows’ location available for reuse

- Strategy for large tables: better pay a little frequently than a lot later on.
- Reduce ***autovacuum_vacuum_scale_factor***
- Reduce ***autovacuum_vacuum_cost_delay***
- Increase ***maintenance_work_mem***
- Increase ***max_parallel_maintenance_workers***
- Proactively run vacuum nightly

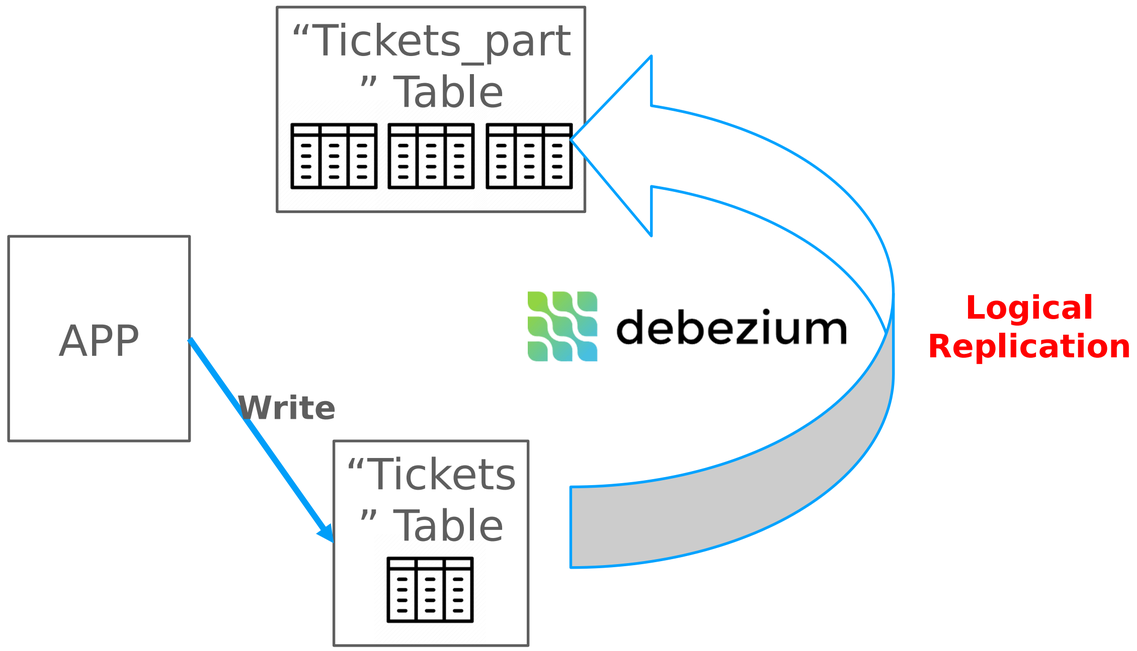
Table Partitioning



- The partition key and the number of partitions (here 128) are critical to get right.
- How did we do it “live” ?

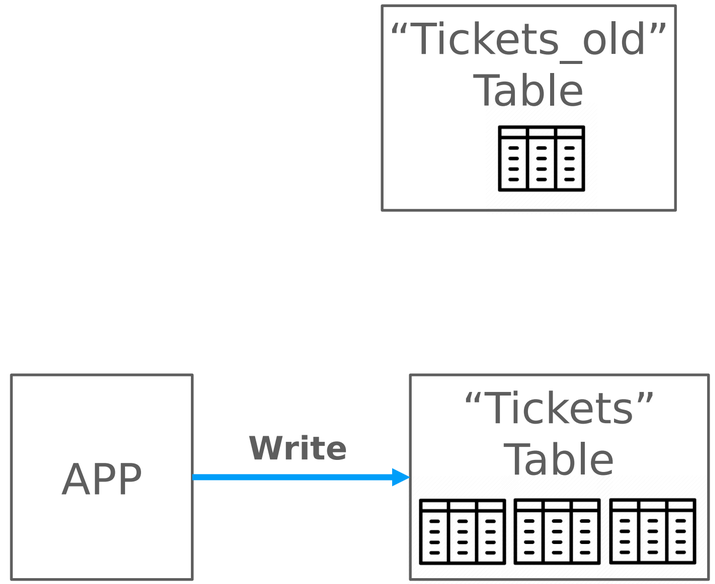
“Live” Table Partitioning

Debezium backfills the new partitioned table and uses logical replication to keep the 2 copies in sync



Note: no App code change, no double write

Short “downtime” and table rename/swap



Issue 3: Locks

- 8 different table-level locks
- Devs regularly need to make schema changes
- Some recipes are well-known: create/drop index concurrently, add NOT NULL field with a DEFAULT
- Some less so: adding a FK constraint in 2 steps with NOT VALID then VALIDATE CONSTRAINT

- Solutions?
  1. CI job with a Linter ?
  2. Teaching and Code reviews

- Not enough: a very short AccessExclusive lock can wait for a long SELECT statement and block all READ/WRITE to a single table
- => Run “migration” with lock_timeout=“10s”

ALTER TABLE

ALTER TABLE — change the definition of a table

Synopsis

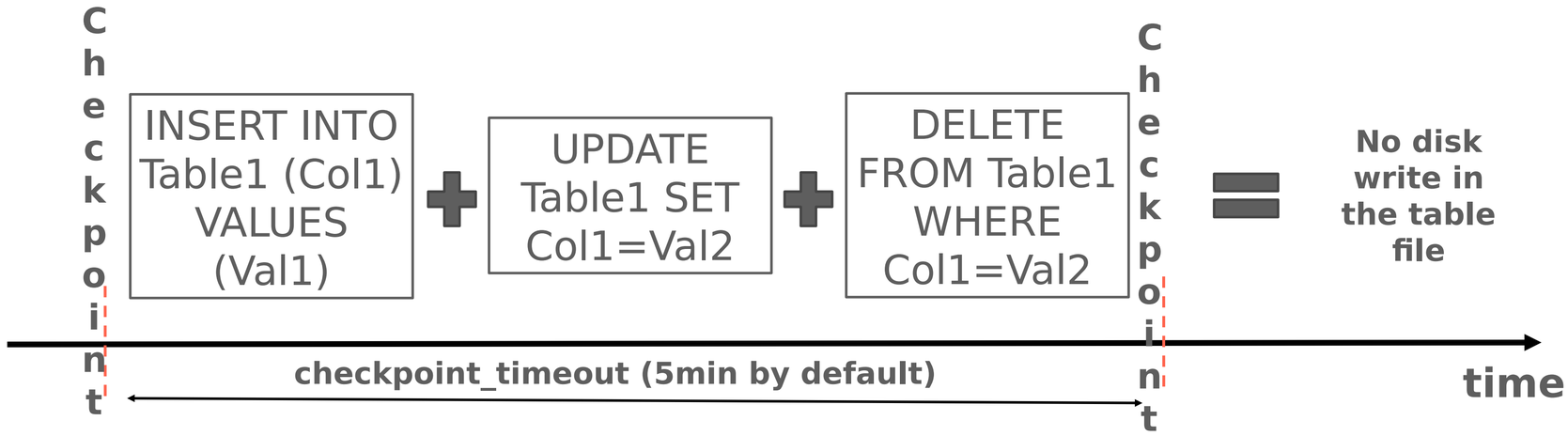
```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema
ALTER TABLE ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]
    SET TABLESPACE new_tablespace [ NOWAIT ]
ALTER TABLE [ IF EXISTS ] name
    ATTACH PARTITION partition_name { FOR VALUES partition_bound_spec | DEFAULT }
ALTER TABLE [ IF EXISTS ] name
    DETACH PARTITION partition_name [ CONCURRENTLY | FINALIZE ]

where action is one of:

    ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
    ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING expression ]
    ALTER [ COLUMN ] column_name SET DEFAULT expression
    ALTER [ COLUMN ] column_name DROP DEFAULT
    ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
    ALTER [ COLUMN ] column_name DROP EXPRESSION [ IF EXISTS ]
    ALTER [ COLUMN ] column_name ADD GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ ( sequence_options ) ]
    ALTER [ COLUMN ] column_name { SET GENERATED { ALWAYS | BY DEFAULT } | SET sequence_option | RESTART [ [ WITH ] restart ] } [...]
    ALTER [ COLUMN ] column_name DROP IDENTITY [ IF EXISTS ]
    ALTER [ COLUMN ] column_name SET STATISTICS integer
    ALTER [ COLUMN ] column_name SET ( attribute_option = value [, ... ] )
    ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
    ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
    ALTER [ COLUMN ] column_name SET COMPRESSION compression_method
    ADD table_constraint [ NOT VALID ]
    ADD table_constraint_using_index
    ALTER CONSTRAINT constraint_name [ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
    VALIDATE CONSTRAINT constraint_name
    DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
    DISABLE TRIGGER [ trigger_name | ALL | USER ]
    ENABLE TRIGGER [ trigger_name | ALL | USER ]
    ENABLE REPLICA TRIGGER trigger_name
    ENABLE ALWAYS TRIGGER trigger_name
    DISABLE RULE rewrite_rule_name
    ENABLE RULE rewrite_rule_name
    ENABLE REPLICA RULE rewrite_rule_name
    ENABLE ALWAYS RULE rewrite_rule_name
    DISABLE ROW LEVEL SECURITY
```

Issue 4: Forced Checkpoints

- What are checkpoints again?
- Checkpoints create a lot of IO disk writes
- Balance between “WAL file not too big” (recovery time) and “saving IOPS”



- If the current WAL file gets bigger than “max_wal_size”, a checkpoint will be forced before checkpoint_time has elapsed.

```
postgres=# select checkpoints_timed, checkpoints_req, 100*checkpoints_req/(checkpoints_timed+checkpoints_req) as ratio_of_forced from pg_stat_bgwriter;
 checkpoints_timed | checkpoints_req | ratio_of_forced
-------------------+-----------------+-----------------
             12402 |             922 |               6
(1 row)
```

Checkpoint tuning

1. Increase “checkpoint_timeout” from 5min (default) to 15min
2. Measure how much WAL grows in 15min

```
postgres@pg-main-0:/$ psql -c "SELECT pg_current_wal_insert_lsn();" && sleep $((15*60)) && psql -c "SELECT pg_current_wal_insert_lsn();"
 pg_current_wal_insert_lsn
---------------------------
 4646/D5865988
(1 row)

 pg_current_wal_insert_lsn
---------------------------
 4647/831335A8
(1 row)

postgres@pg-main-0:/$ psql -c "SELECT pg_size_pretty(pg_wal_lsn_diff('4647/831335A8','4646/D5865988'));"
 pg_size_pretty
----------------
 2777 MB
(1 row)
```

4. Set that as “max_wal_size” value (maybe add 20% to be safe)
5. Forcefully kill PG (SIGKILL) and measure time to recovery

```
15:32:36.021 UTC [13] LOG:  database system was not properly shut down; automatic recovery in progress
15:32:36.035 UTC [13] LOG:  redo starts at 22/3E5CB440
15:32:36.036 UTC [13] LOG:  invalid record length at 22/3E5E7530: wanted 24, got 0
15:32:36.036 UTC [13] LOG:  redo done at 22/3E5E74F8 system usage: CPU: user: 0.00 s, system: 0.00 s, el
15:32:36.060 UTC [13] LOG:  checkpoint starting: end-of-recovery immediate
15:32:36.447 UTC [13] LOG:  checkpoint complete: wrote 60 buffers (0.0%); 0 WAL file(s) added, 0 removed
s, average=0.005 s; distance=112 kB, estimate=112 kB
15:32:36.461 UTC [1] LOG:  database system is ready to accept connections
```

6. If “too long” reduce checkpoint_timeout and max_wal_size

Issue 5: large JSONB Columns

- JSONB type is great !
- PG could feel like a document store with ACID guaranties
- BUT!
- No column statistics
- Large storage footprint:
  - Duplicated key names
  - Out-of-line storage (a.k.a TOAST)
    - No Partial updates
    - No Partial read

Issue 6: Running PG in Kubernetes

- The setup (per shard):
  - 1 StatefulSet for the PG primary with exactly 1 replica
  - 1 StatefulSet for the PG Hot Standbys with N replicas

```bash
#!/usr/bin/env bash
set -ueo pipefail

if [ -n "${PRIMARY_HOSTNAME:-}" ]; then
    echo "Setting up postgres replica"
    pg_ctl -D "${PGDATA}" -m fast -w stop
    # remove all data before doing the basebackup
    rm -rf "${PGDATA:?}"/*
    until pg_isready -h "${PRIMARY_HOSTNAME}"; do echo 'waiting for primary'; sleep 1; done
    # do the initial backup from the main server
    PGPASSWORD="$REPLICA_PASSWORD" pg_basebackup -h "${PRIMARY_HOSTNAME}" -D "${PGDATA}" -U "${REPLICA_USER}" --slot="${HOSTNAME}" --create-slot --write-recovery-conf --progress -X stream
    # start postgres again
    pg_ctl -D "$PGDATA" -o "-c listen_addresses='*'" -w start
else
    echo "Setting up postgres main"
    # create replica user
    psql postgres -U "${POSTGRES_USER}" -c "CREATE USER ${REPLICA_USER} REPLICATION LOGIN CONNECTION LIMIT 6 ENCRYPTED PASSWORD '${REPLICA_PASSWORD}'"
    # Allow replica hosts to connect
    echo "host replication ${REPLICA_USER} 0.0.0.0/0 md5" >> "${PGDATA}/pg_hba.conf"
    # Note that the replication slots are created dynamically with the --create-slot option of pg_basebackup
    echo "Finished setting up postgres main"
fi
```

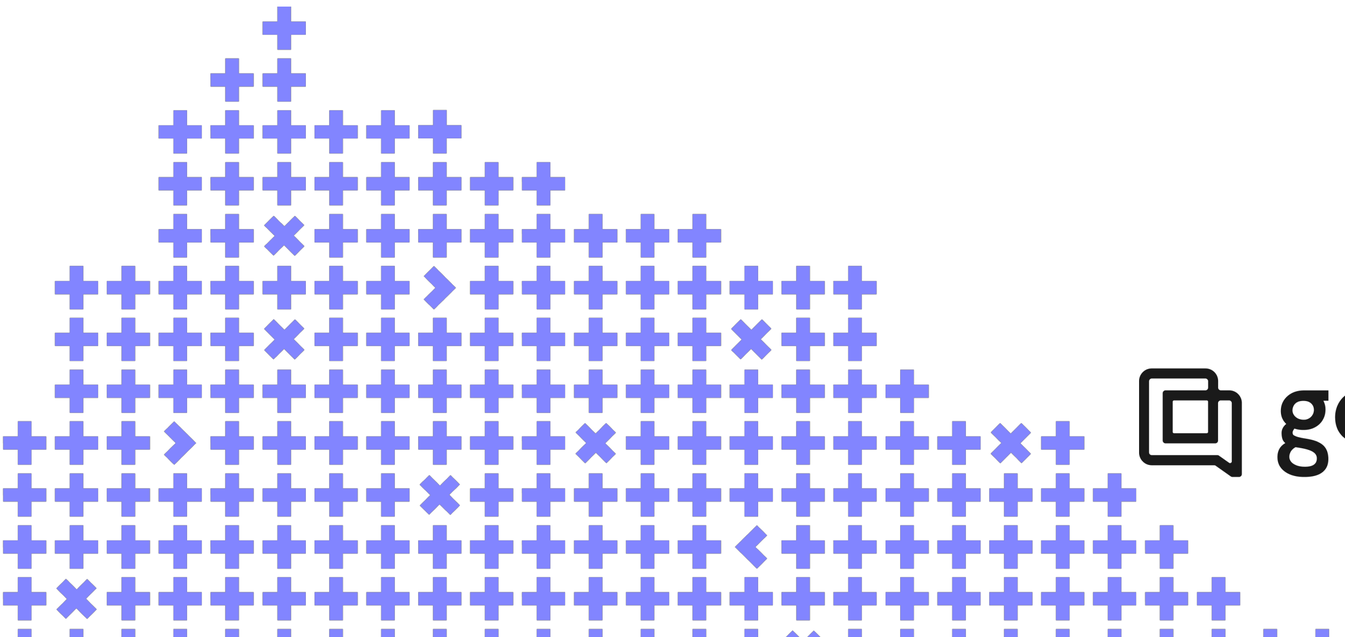

Conclusion

- Start with PG for JSON documents, for time series, for analytics, etc.
- You can go a long way with 1 primary and 1+ RO replicas. (and 512Gb of RAM)
- Get help and more specialized data store later on… but not too late!
- Debezium and CDC is a Swiss army knife

Leave your feedback!

You can rate the talk and give a feedback on what you've liked or what could be improved

я



gorgias

++ High Load Armenia

Co-organizer

Yandex